SSD & FTL从底向上 (P2)

Original Accela Zhao Accela推箱子 2018-01-01

（续前文：SSD & FTL 从底向上 (P1)）

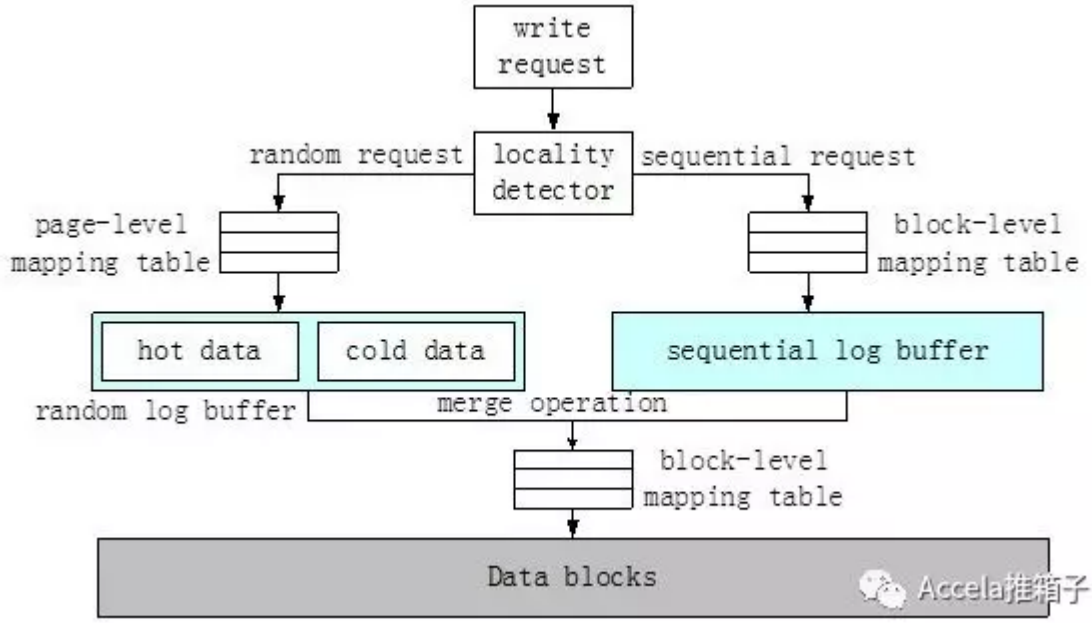
FTL – 混合映射（Hybrid Mapping）

从上面的种种功能中，可以看到FTL的实现首先有两大问题：一是如何解决逻辑地址到物理地址的映射；二是在用户指定的地址已经被写过数据的情况下，如何快速地吸收新的写，而不需要等待较耗时间的块擦除操作。

地址映射，最直接的方法是块级映射（Block-level address mapping），即元数据仅记录到块的粒度。这样实现简单直接，只需要很少的元数据。但映射粒度太粗，如果用户所写的页地址和物理实际的页地址不一致，块级映射无法表达。

另一个方法页级映射（Page-level address mapping），即元数据记录页到页的映射关系，粒度细，可以精细地管理分配。但问题是，所需元数据体积过大。例如500GB的SSD，每页4KB，需要125M地址项；如果每个地址映射表项占用4B，则一共占用500MB。对FTL硬件来说，所需内存空间过大。

实践中真正被使用的方法是混合映射（Hybrid Mapping），它混合了块级映射和页级映射。并不只是简单地混合两者，还结合了避免擦除、快速吸收新写的方法。



[Source: AlanWu 博客 - 神秘的 FlashTranslation Layer] (http://blog.51cto.com/alanwu/1427101)

经典的方法如上图所示。SSD中的块被分为两种，一直是数据块（Data block），用于存放用户数据；另一种被称作日志块（Log block），或者其它名字如Buffer block / Log buffer等。日志块对应上图的蓝色部分。新的用户写不会被直接存放在数据块中，即不是Write-in-place的，而是首先写入日志块中。当日志块满后，进行合并（Merge）操作，将缓存在日志块中的用户数据放入它们应属位置的数据块；这一步称作垃圾回收（GC）。SSD中，绝大部分区域都是数据块，少部分区域被用作日志块；一般而言日志块越多，快速吸收写性能越好；而这也是很多商用SSD实际可用存储空间比总体空间小的原因（Over-provisioning）。

这种非Write-in-place的设计与LSM-tree或者日志结构（Log-structured）或者Append-only等异曲同工，而它们也都需要重点对垃圾回收进行优化。本质上它们是优化写操作而牺牲读操作的设计；因为对用户写而言连续的地址，可能被存储在日志块和数据块的随机位置，读操作变成了不连续的。但是，对SSD而言，随机读性能优异，与顺序读差别不大，所以正是使用此种设计的佳处。而磁盘配LSM-Tree数据结构时，往往需要配内存缓存或内存中的表以减轻读的压力，而磁盘上则需要定时合并（Compact）数据以重排乱序。

继续对上图的讲解。数据块中，页级地址与用户所指定的页级地址一致，对数据块只需要保存块级地址映射元数据即可。至少数据块内，用户的连续地址数据也被物理地连续存储，对

读缓存预取也有好处。

日志块中，同一块内混合了不同来源的用户写，可能是乱序的，则有使用页级地址映射的需要。实际上，改进后的FTL算法能够区分用户正在进行顺序写还是随机写，对此采用不同的策略。区分的方法基本上是机会主义的，统计写入块的平均大小，以及各次地址是否是连续的；在后面的论文中有更详细的描述。顺序写进入上图右侧的顺序日志块，非顺序写走左侧随机日志块。因为是顺序写，FTL可以让块页的物理地址与逻辑地址保持一致，因此只需要块级地址映射即可。而更为关键的是，垃圾回收合并时代价很小，Switch-Merge即可（后面讲）。上图左侧对随机写的处理中，还进一步区分了冷热数据；论文中有更详细的讲解，区分冷热数据可以进一步减小垃圾回收的合并开销。

下面继续讲垃圾回收。垃圾回收操作主要是合并（Merge），即把写满的日志块中的数据放到数据块中。对应数据块中可能已有数据，需要把它们挪位置，或者与日志块中的新写合并更新。合并操作大致有三种：

[Source: LAST -Locality-Aware Sector Translation] (http://yourcmc.ru/wiki/images/d/d2/02-lee-last-usenix09.pdf)

- Switch-Merge：被合并的数据块里没有需要保留的数据，而日志块中的页地址也不需要再挪动。合并时，只需要交换对应的日志块和数据块；修改地址映射表，而不需要真实地重写数据。Switch-Merge的开销是最小的。例如上文中的顺序日志块，往往总能Switch-Merge，从而减小垃圾回收的开销；这是区分用户顺序写和随机写的原因。

- Partial-Merge：与Switch-Merge类似，往往来自于顺序日志块的合并，最终的数据块和日志块直接交换即可。但是，数据块上仍顺序地比日志块多出一部分数据，仍被需要；FTL把它拷贝给日志块对应的页。总体上，Partial-Merge开销较小。

- Full-Merge：日志块与数据块各自包含一部分需要的数据，于是需要把数据合并重写到一个新的块上去，从而释放日志块和数据块。重写数据开销较大，而数据重写也是写放大的主要原因之一。Full-Merge主要来自与随机写，即随机日志块。

上文基本上就是FTL的主要工作原理；论文中和实际工业产品在此基础上会有更多优化。解决了逻辑地址到物理地址的映射，和在避免擦除延时的同时快速吸收新写的问题，FTL的其它功能就可以被安装在这个骨架上。例如坏块管理可以由地址映射逻辑实现，磨损平衡体现在如何选择和分配日志块、数据块中，减轻写放大需要优化垃圾回收、减少Full-Merge等。

总之，一个实用的推论是，顺序写对SSD也有利，因为其大幅减轻垃圾回收代价。

FTL - 混合映射（续）

解开了FTL的神秘面纱后，下面列了一些我找出来的有代表性的论文。它们基本发表于2002至2008年，基本全部出自于三星和韩国大学。

[A Space-efficient Flash Translate Layer for CompactflashSystems](https://pdfs.semanticscholar.org/e0a1/546f56b68ebfcc5f7237c073d6186188f192.pdf)

这篇论文是混合映射的开山之作。它提出了保留少量日志块，用作写缓存，以吸收覆写（Overwirte）。（与“覆写”相对的是写在没有数据的块上的写，不需要擦除操作。）

[A reconfigurable FTL (flash translation layer) architecturefor NAND flash-based applications](http://csl.skku.edu/papers/tecs08.pdf)

这篇论非常“慈善”，在背景和相关工作部分，它详尽地介绍了Flash的各种概念，和FTL混合映射的发展历史。从中可以看出一系列论文是如何一步步改进出如今的FTL的。

[FAST: A Log Buffer-Based Flash Translation Layer UsingFully-Associative Sector Translation](http://csl.skku.edu/uploads/ICE3028S11/fast-tecs07.pdf)

上面的论文被称作FAST，它基于“A Space-efficient Flash ..”改进，允许一个日志块被全体数据块共享使用（如今看来很正常），而不是“A Space-efficient Flash ..”中的一个日志块只与一个数据块对应（见“A reconfigurable FTL ..”对其的引用评价）。另一方面，FAST提供了一个额外的日志块，专用与顺序写处理，这是顺序日志块的雏形。

[A Superblock-based Flash Translation Layer for NAND FlashMemory] (http://csl.skku.edu/papers/emsoft06.pdf)

这篇论文中的“超级块”，有些类似于Linux的Ext2文件系统中的超级块。每个超级块配有N个数据块和M个日志块，自成一体；即类似Ext2中的多个超级块，每个都自成一体地管理着自己辖区内的元数据如块分配位图、Inode位图、Inode表等，以及数据块。

[LAST: Locality-Aware Sector Translation for NAND FlashMemory-Based Storage Systems](http://yourcmc.ru/wiki/images/d/d2/02-lee-last-usenix09.pdf)

这篇论叫LAST，基于FAST和Superblock做了更多改进。LAST将日志块区分为顺序日志块和随机日志块，随机日志块被按照冷数据和热数据区分，使用不同策略。热数据区的数据，利用时间局部性（Temporal locality）进一步减小Full-Merge的开销，例如将热页聚集在同一个日志块中，等等。对应地，顺序日志块被称作在利用顺序局部性（Sequential locality）。这个论文的设计与文章开头讲述的混合映射（Hybrid Mapping）已经基本一致了。

绕过FTL – Open-ChannelSSD

FTL功能复杂强大，如今在SSD中被（绝对地）主流使用。但与之相应的是FTL其带来的额外开销，黑箱锁住的难以定制的功能。一些要求高度定制化和更低延迟的用户，如大型互联网公司（如下面SDF和LevelDB改造论文由百度参与），可能希望绕开FTL，暴露底层特性，直接操作Flash。FTL SSD被总结有如下问题

- 因为FTL有垃圾回收等背景任务，SSD读写速度可能会难以预料地、时不时地变慢。
- 去掉FTL可以降低访问SSD的延迟。如下面的SDF论文中指出，相比访问裸Flash的带宽，FTL版的读带宽只有73%～81%，写带宽只有41%～51%。
- 应用应该能够按照自身特性，定制化地优化Flash操作，绕过FTL。下面的AMF论文对其有更多描述。

Open-Channel SSD指SSD应该将内部的Flash通道（Channel）、物理地址映射、垃圾回收、块擦除、覆写等功能暴露给应用使用。现在主要是一个方向，还没有统一的方案。

百度在Open-Channel SSD方面有不少些研究。下面论文在LevelDB上进行改造，底层使用百度研发的SDF Open-Channel SSD。改进后的LevelDB能够充分利用Open-Channel SSD暴露出来的多个并行的Flash通道，并且优化了并发IO的调度和分发策略。相比跑在普通SSD上的原装LevelDB，吞吐量能够提高4x。

[An Efficient Design and Implementation of LSM-Tree basedKey-Value Store on Open-Channel SSD](http://www.ece.eng.wayne.edu/~sjiang/pubs/papers/wang14-LSM-SDF.pdf)

百度的SDF论文如下。SDF是硬件特殊定制的SSD，将Flash通道暴露给应用，并且消除Over-provisioning（商用SSD常保留一部分存储空间，即预留日志块以高速吸收新写），允许应用充分利用Flash多通道的天然并行，调度数据访问。已有3000多SDF部署在百度产线（论文发表时间为2014年），用于支持网页和图片库服务。SDF能够提供裸Flash带宽的95%和存储容量的99%（上文中有列FTL SSD的带宽），相对于普通商用硬件（Commodity hardware）减少每GB成本的50%。

[SDF: Software-Defined Flash for Web-Scale Internet StorageSystems](https://pdfs.semanticscholar.org/6197/7858b3eea4f5a6d81393301e7298ade7a2d8.pdf)

此外，开源社区经典的RocksDB也对Open-Channel SSD做过一番优化。主要在控制数据分布（placement），利用Flash的并行性，调度垃圾回收，减少Over-provisioning，控制IO调度等方面。

[Optimizing RocksDB for Open-Channel SSDs](http://www.slideshare.net/JavierGonzlez49/optimizing-rocksdb-for-openchannel-ssds)

前文中提到，Linux 4.4内核中新整合了对Open-Channel SSD的支持，即LightNVM。其像NVMe（NVM Express）协议靠拢，全栈定义了接口规范、操作系统支持、用户控件管理工具、IO库等。另外，在2017年的FAST存储会议上，LightNVM也发了文章（看样子Open-Channel SSD发展挺快）。

[Wikipedia: LightNVM](https://en.wikipedia.org/wiki/Open-channel_SSD)

[LightNVM site](http://lightnvm.io/)

[FAST17: LightNVM: The Linux Open-Channel SSD Subsystem](https://www.usenix.org/conference/fast17/technical-sessions/presentation/bjorling)

最后，还有一篇AMF论文，采取相对中性一些的策略。它定制了一个比FTL简单些的Flash控制器硬件，对应用暴露块IO接口（Block IO Interface），并且要求覆写前必须擦除（erase-before-overwrite）。逻辑地址到物理地址的映射、坏块管理、磨损平衡等仍然由AMF硬件控制器管理（垃圾回收不由AMF管理，因为应用自己控制擦除）。另外，AMF也允许应用对来自多个Flash通道的并行能力进行利用。

[Application-Managed Flash](http://people.csail.mit.edu/ml/pubs/fast2016.pdf)

综上，我们对SSD的内部实现和特性已经有了相当的了解。分布式存储系统为了适应SSD的许多新特性，有许多设计方面的优化和变迁。毕竟，相对于近年来存储和网路性能的大幅度

提升，CPU因为摩尔定律到达尽头和频率、发热限制而渐渐跟不上。后文中将会有更多解析。

（未完待续……）